Attribute-Based Encryption with Fast Decryption

Susan Hohenberger and Brent Waters

[1] Johns Hopkins University, susan@cs.jhu.edu
[2] University of Texas at Austin, bwaters@cs.utexas.edu

Abstract. Attribute-based encryption (ABE) is a vision of public key encryption that allows users to encrypt and decrypt messages based on user attributes. This functionality comes at a cost. In a typical implementation, the size of the ciphertext is proportional to the number of attributes associated with it and the decryption time is proportional to the number of attributes used during decryption. Specifically, many practical ABE implementations require one pairing operation per attribute used during decryption.

This work focuses on designing ABE schemes with fast decryption algorithms. We restrict our attention to expressive systems *without* system-wide bounds or limitations, such as placing a limit on the number of attributes used in a ciphertext or a private key. In this setting, we present the first key-policy ABE system where ciphertexts can be decrypted with a constant number of pairings. We show that GPSW ciphertexts can be decrypted with only 2 pairings by increasing the private key size by a factor of $|\Gamma|$, where Γ is the set of distinct attributes that appear in the private key. We then present a generalized construction that allows each system user to independently tune various efficiency tradeoffs to their liking on a spectrum where the extremes are GPSW on one end and our very fast scheme on the other. This tuning requires no changes to the public parameters or the encryption algorithm. Strategies for choosing an individualized user optimization plan are discussed. Finally, we discuss how these ideas can be translated into the ciphertext-policy ABE setting at a higher cost.

1 Introduction

Attribute-based encryption (ABE) [18] is an expansion of public key encryption that allows users to encrypt and decrypt messages based on user attributes. In a *key-policy* ABE (KP-ABE) system, an encrypted message can be tagged with a set of attributes, such as tagging an email with the metadata "from: Alice", "to: IACR board", "subject: voting", "date: October 1, 2012", etc. The master authority for the system can issue private decryption keys to users including an access policy, such as giving to Bob a decryption key that enables him to decrypt any ciphertexts that satisfy "to: Bob" OR ("to: IACR board" AND (January 1, 2011 $\leq$"date" $\leq$ December 31, 2012)).

This access control functionality can be very powerful, but also costly. In this work, we focus on the cost of decryption. In many key-policy ABE systems, such as that of Goyal, Pandey, Sahai and Waters (GPSW) [13], the decryption algorithm requires one pairing for each attribute used during decryption. (Encryption does not require any pairings, and is thus already fast by comparison.)

It seems conceivable that one might reduce the cost of decryption by making tradeoffs elsewhere. One tradeoff we allow ourselves in this work is to increase the private key size, although we ideally want to limit any increase as much as possible. We do not, however, consider tradeoffs that increase the ciphertext size or that place any limitations on how the ABE system can be used. That is, we focus on fast decryption for the most general setting possible – an expressive, large-universe system, where there are no bounds on, say, the number of attributes that can appear in a ciphertext or private key. While good progress has been made on efficient ABE in "bounded settings", as we discuss shortly, our focus is to develop techniques for improving efficiency in the most general setting and for applications where it is infeasible to trade system-wide usability for performance.

Our Contributions We present the first expressive and "unbounded" key-policy ABE (KP-ABE) system in a pairing setting, which requires only a constant number of pairings to decrypt any ciphertext. It builds upon the GPSW system [13]. It reduces the decryption requirements to two pairings and two exponentiations (these exponents are 1 or 0 if the access policy is a boolean formula), while increasing the number of multiplications by a factor of $|\Delta|$, where Δ is the set of distinct attributes used during decryption. It also increases the private key size by a factor of $|\Gamma|$, where Γ is the set of distinct attributes used in the private key.

We discuss several variants of this system, including a method for reducing the ciphertext size from $O(|\Delta|)$ to three group elements (in the small universe setting only) at the cost of larger private keys. We also discuss the difficulties of achieving fast decryption for ciphertext-policy ABE (CP-ABE) systems in an unbounded setting, as well as progress in the bounded setting.

In Section 5, we present generalized decryption and "key storage" algorithms that allow each system user to independently tune various efficiency tradeoffs to their liking on a spectrum that ranges from GPSW (shorter keys, slow decryption) on one end to our main KP-ABE construction (longer keys, fast decryption) on the other. This tuning requires no changes to the setup, encryption or key generation algorithms of our base construction. Rather, it is managed transparently by each user, who can choose among many possible decryption algorithms where the amount of her private key that she needs to securely store scales accordingly. We conclude with some strategies for choosing an individualized user optimization plan.

1.1 Related Work

To our knowledge, the only prior work to achieve constant pairings in decryption for an expressive KP-ABE system is that of Attrapadung, Herranz, Laguillaumie, Libert, de Panafieu and Ráfols [3, 2]. The goal of their work was on achieving short ciphertexts. They were able to achieve very short ciphertexts (constant number of group elements) using novel applications of aggregation techniques that have roots in those used to achieve hierarchical identity-based encryption [14, 11] with constant size ciphertexts by Boneh, Boyen and Goh [6] and those used to achieve practical broadcast encryption by Boneh, Gentry and Waters [8]. Our ideas begin at this starting point as well. However, Attrapadung et al. brought in an inner product instance as a base building block which *fundamentally* demands a bound, n, on the maximum number of attributes that can appear in a ciphertext. This choice of n forces a tradeoff between flexibility and performance.

One needs to set n high to cover the "worst" case even if the typical encryption uses far fewer attributes. Unfortunately, their private key size blows up by a factor of n, whereas our key size only increases by a factor of the number of distinct attributes used in *that particular* key. Moreover, for a ciphertext that encrypts with $|S|$ attributes, they have an added cost of $|S|$ exponentiations during decryption. This could be very costly in some situations. Suppose Alice has the key policy (A_1 AND A_2) and receives a ciphertext with many attributes $A_1, A_2, \ldots, A_{1000}$. In their system, Alice must do 1000 exponentiations, which is actually much worse than if she was only required to do one pairing per attribute used in decryption.[3]

This work aims to avoid these "worst case" penalties. In particular, we want that a user's key size be related to the complexity of her key. In their scheme, it grows with n. In decryption, the computational cost should be related to how many rows of the LSSS one must use. In their scheme, the number of exponentiations grows with the number of attributes in the ciphertext. This is a tradeoff relative to GPSW [13]. Finally, in their scheme, the number of multiplications is roughly $|I| \cdot |S|$, the number of rows used in decryption times the number of attributes in a ciphertext. Ours is $|I| \cdot |\Delta|$, the number of rows used in decryption times the number of distinct attributes used in decryption. Note that $|\Delta| \leq |S|$.

While we have pointed out some of the areas for improvement in Attrapadung et al. for comparison's sake, we do wish to stress that this was a pioneering work that showed that short ciphertexts and fast decryption was possible at all for KP-ABE. We also refer the reader to that work for an excellent summary of the efficiency of prior ABE schemes. Our work compliments theirs by taking the study of fast decryption for ABE into the unbounded realm with tighter growth.

We note that Identity-Based Encryption [19, 7, 10] was an early forerunner of ABE and several technique from IBE impact ABE systems. Finally, in this

[3] The polynomial related to Y in their construction grows with the number of attributes in the ciphertext.

work, we only consider selective security [9], however, we believe our techniques could apply to more recent systems proven adaptively secure using dual system encryption [15, 17].

2 Background

This section covers background information. We make use of the standard definitions for access structures and linear secret sharing schemes (LSSS), as well as the conventions and notation for these employed in several prior ABE works. These are included in Appendix A for reference.

2.1 Definitions of Security for Key Policy ABE Schemes

Definition 1 (KP-ABE Algorithm Specification). *A key-policy attribute-based encryption system for message space $\mathcal{M}$ and access structure space $\mathcal{G}$ is a tuple of the following algorithms:*

Setup$(\lambda, U) \rightarrow (\mathrm{PK}, \mathrm{MK})$**.** *The setup algorithm takes as input a security parameter λ and a universe description U, which defines the set of allowed attributes in the system. It outputs the public parameters* PK *and the master secret key* MK*.*

Encrypt$(\mathrm{PK}, M, S) \rightarrow \mathrm{CT}$**.** *The encryption algorithm takes as input the public parameters* PK*, a message M and a set of attributes S and outputs a ciphertext* CT *associated with the attribute set.*

KeyGen$(\mathrm{MK}, \mathbb{A}) \rightarrow \mathrm{SK}$**.** *The key generation algorithm takes as input the master secret key* MK *and an access structure $\mathbb{A}$ and outputs a private key* SK *associated with the attributes.*

Decrypt$(\mathrm{SK}, \mathrm{CT}) \rightarrow M$**.** *The decryption algorithm takes as input a private key* SK *associated with access structure $\mathbb{A}$ and a ciphertext* CT *associated with attribute set S and outputs a message M if S satisfies $\mathbb{A}$ or the error message $\perp$ otherwise.*

The correctness property requires that for all sufficiently large $\lambda \in \mathbb{N}$, all universe descriptions U, all $(\mathrm{PK}, \mathrm{MK}) \in \mathsf{Setup}(\lambda, U)$*, all $S \subseteq U$, all* $\mathrm{SK} \in \mathsf{KeyGen}(\mathrm{MK}, \mathbb{A})$*, all $M \in \mathcal{M}$, all $\mathbb{A} \in \mathcal{G}$ and all* $\mathrm{CT} \in \mathsf{Encrypt}(\mathrm{PK}, M, S)$*, if S satisfies $\mathbb{A}$, then* $\mathsf{Decrypt}(\mathrm{SK}, \mathrm{CT})$ *outputs M.*

Security Model for KP-ABE Let $\Pi = (\mathsf{Setup}, \mathsf{Encrypt}, \mathsf{KeyGen}, \mathsf{Decrypt})$ be a KP-ABE scheme for message space $\mathcal{M}$ and access structure space $\mathcal{G}$, and consider the following experiment for an adversary $\mathcal{A}$, parameter λ and attribute universe U:

The KP-ABE experiment $\mathsf{KP\text{-}ABE\text{-}Exp}_{\mathcal{A},\Pi}(\lambda, U)$:

Setup. The challenger runs the Setup algorithm and gives the public parameters, PK to the adversary.

Phase 1. The challenger initializes an empty table T, an empty set D and an integer counter $j = 0$. Proceeding adaptively, the adversary can repeatedly make any of the following queries:

- Create($\mathbb{A}$): The challenger sets $j := j + 1$. It runs the key generation algorithm on $\mathbb{A}$ to obtain the private key SK and stores in table T the entry $(j, \mathbb{A}, \text{SK})$.
  Note: Create can be repeatedly queried with the same input.
- Corrupt(i): If there exists an i^{th} entry in table T, then the challenger obtains the entry $(i, \mathbb{A}, \text{SK})$ and sets $D := D \cup \{\mathbb{A}\}$. It then returns to the adversary the private key SK. If no such entry exists, then it returns $\perp$.
- Decrypt(i, CT): If there exists an i^{th} entry in table T, then the challenger obtains the entry $(i, \mathbb{A}, \text{SK})$ and returns to the adversary the output of the decryption algorithm on input (SK, CT). If no such entry exists, then it returns $\perp$.

Challenge. The adversary submits two equal length messages M_0 and M_1. In addition the adversary gives a set of attributes S^* such that for all $\mathbb{A} \in D$, the set S^* does not satisfy the access structure $\mathbb{A}$. The challenger flips a random coin b, and encrypts M_b under S^*. The resulting ciphertext CT^* is given to the adversary.

Phase 2. Phase 1 is repeated with the restrictions that the adversary cannot

- trivially obtain a private key for the challenge ciphertext. That is, it cannot issue a Corrupt query that would result in an access structure $\mathbb{A}$ which S^* satisfies being added to D.
- issue a decryption query on the challenge ciphertext CT^*.

Guess. The adversary outputs a guess b' of b. The output of the experiment is 1 if and only if $b = b'$.

Definition 2 (KP-ABE Security). *A KP-ABE scheme Π is CCA-secure (or secure against* chosen-ciphertext attacks*) for attribute universe U if for all probabilistic polynomial-time adversaries $\mathcal{A}$, there exists a negligible function negl such that:*

$$\Pr[\mathsf{KP\text{-}ABE\text{-}Exp}_{\mathcal{A},\Pi}(\lambda, U) = 1] \leq \frac{1}{2} + negl(\lambda).$$

CPA Security. We say that a system is *CPA*-secure (or secure against *chosen-plaintext attacks*) if we remove the Decrypt oracle in both Phase 1 and 2.

Selective Security. We say that a system is *selectively* secure if we add an Init stage before Start where the adversary outputs the challenge attribute set S^* (instead of waiting until Challenge).

2.2 Bilinear Maps

Let $\mathbb{G}$ and $\mathbb{G}_T$ be two multiplicative cyclic groups of prime order p. Let g be a generator of $\mathbb{G}$ and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be a bilinear map with the properties:

(1) Bilinearity: for all $u, v \in \mathbb{G}$ and $a, b \in \mathbb{Z}_p$, we have $e(u^a, v^b) = e(u, v)^{ab}$ and (2) Non-degeneracy: $e(g, g) \neq 1$. We say that $\mathbb{G}$ is a bilinear group if the group operation in $\mathbb{G}$ and the bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ are both efficiently computable. We now state an assumption used in the constructions.

Definition 3 (Decisional BDHE [6]). *Let $a, s \in \mathbb{Z}_p$ be chosen at random and g be a generator of group $\mathbb{G}$ of prime order $p \in \Theta(2^\lambda)$. The decisional q-BDHE assumption is that all probabilistic polynomial-time algorithms $\mathcal{A}$ given the vector $\boldsymbol{y}=$*

$$\mathbb{G}, p, g, g^s, g^a, \ldots, g^{(a^q)}, g^{(a^{q+2})}, \ldots, g^{(a^{2q})}$$

have an advantage negligible in λ of distinguishing $e(g, g)^{a^{q+1}s} \in \mathbb{G}_T$ from a random element in $R \in \mathbb{G}_T$. The advantage of $\mathcal{A}$ is defined as

$$\left| \Pr\left[\mathcal{A}(\boldsymbol{y}, e(g, g)^{a^{q+1}s}) = 0\right] - \Pr\left[\mathcal{A}(\boldsymbol{y}, R) = 0\right] \right|$$

where the probability is taken over the random choice of a, s in $\mathbb{Z}_p$, R in $\mathbb{G}_T$ and the generator g, and the random bits consumed by $\mathcal{A}$.

3 ABE with Fast Decryption

3.1 The Base Construction: Small Universe KP-ABE

We first describe a system for a small universe U of attributes, where $|U|$ is a polynomial in 1^λ, and the attributes are the integers $1, \ldots, U$. Subsequently, we will describe how to alter this construction to accommodate a large universe $U = \{0, 1\}^*$ of attributes in the random oracle model. When we refer to our base construction in a setting where large universes are assumed, we mean this close variant. The message space is $\mathbb{G}_T$.

Setup$(\lambda, U) \to (\mathrm{PK}, \mathrm{MK})$. The setup algorithm first chooses a bilinear group $\mathbb{G}$ of prime order $p \in \Theta(2^\lambda)$. It selects a random generator $g \in \mathbb{G}$. It next selects random values $h_1, \ldots, h_{|U|} \in \mathbb{G}$ and $\alpha \in \mathbb{Z}_p$. It then sets the keys as:

$$\mathrm{PK} = (\mathbb{G}, p, g, e(g, g)^\alpha, h_1, \ldots, h_{|U|}), \quad \mathrm{MK} = (\mathrm{PK}, \alpha).$$

Encrypt$(\mathrm{PK}, M, S) \to \mathrm{CT}$. The encryption algorithm takes as input the public parameters PK, a message $M \in \mathbb{G}_T$ to encrypt, and a set of attributes S. It chooses a random $s \in \mathbb{Z}_p$. The ciphertext is published as $\mathrm{CT} = (S, C, \hat{C}, \{C_x\})$ where

$$C = M \cdot e(g, g)^{\alpha s}, \ \hat{C} = g^s, \ \{C_x = h_x^s\}_{x \in S}.$$

KeyGen$(\mathrm{MK}, \mathbb{A}) \to \mathrm{SK}$. The key generation algorithm takes as input the master secret key and an LSSS access structure (W, ρ). Let W be an $\ell \times n$ matrix. The function ρ associates rows of W to attributes. Let Γ denote the set of distinct attributes the appear in the access structure matrix W; that is, $\Gamma = \{d : \exists i \in [1, \ell], \rho(i) = d\}$. The algorithm first chooses a random vector $\boldsymbol{v} = (\alpha, y_2, ..., y_n) \in \mathbb{Z}_p^n$. These values will be used to share the master secret α. For $i = 1$ to ℓ, it calculates $\lambda_i = \boldsymbol{v} \cdot W_i$, where W_i is the vector corresponding to the ith row of W. In addition, the algorithm chooses random $r_1, \ldots, r_\ell \in \mathbb{Z}_p$. It sets the private key SK as:

$$\mathrm{PK}, \quad (D_1 = g^{\lambda_1} \cdot h_{\rho(1)}^{r_1},\ R_1 = g^{r_1},\ \forall d \in \Gamma/\rho(1),\ Q_{1,d} = h_d^{r_1}),\ \ldots\ ,$$
$$(D_\ell = g^{\lambda_\ell} \cdot h_{\rho(\ell)}^{r_\ell},\ R_\ell = g^{r_\ell},\ \forall d \in \Gamma/\rho(\ell),\ Q_{\ell,d} = h_d^{r_\ell}).$$

In our notation above, we slightly abuse the set minus notation, and by Γ/x, where Γ is a set and x is a single element, we mean $\Gamma/\{x\}$; i.e., the set Γ with the element x removed if present.

These keys contain GPSW [13] keys with the addition of the "helper values" $Q_{i,d}$. The key size is proportional to $|\Gamma| \cdot \ell$, which is the number of distinct attributes that appear in the access matrix times the number of rows in the matrix. Since $|\Gamma| \leq \ell$, we have $|\Gamma| \cdot \ell \leq \ell^2$.

Decrypt$(\mathrm{SK}, \mathrm{CT}) \to M$. The decryption algorithm takes as input a key $\mathrm{SK} = (\mathrm{PK}, (D_1, R_1, \{Q_{1,d}\}), \ldots, (D_\ell, R_\ell, \{Q_{\ell,d}\}))$ for access structure (W, ρ) and a ciphertext $\mathrm{CT} = (C, \hat{C}, \{C_x\}_{x \in S})$ for set S. Let W be an $\ell \times n$ matrix. The function ρ associates rows of W to attributes. If S does not satisfy the access structure, it outputs $\perp$. Suppose that S satisfies the access structure and let $I \subseteq \{1, 2, \ldots, \ell\}$ be a set of indices and $\{\omega_i\}_{i \in I} \in \mathbb{Z}_p$ be a set of constants such that:

1. For all $i \in I$, $\rho(i) \in S$.
2. $\sum_{i \in I} \omega_i \cdot W_i = (1, 0, 0, \ldots, 0)$.

We then define $\Delta = \{x : \exists i \in I,\ \rho(i) = x\}$. That is, I is the set of indices corresponding to the rows used in one possible way to decrypt the ciphertext and Δ is the set of distinct attributes associated with these rows. In general, there can be multiple such I that satisfy the above constraints. Typically, one will wish to minimize the size of I. Note that $\Delta \subseteq S$, where S is the attributes used to encrypt the ciphertext, and $\Delta \subseteq \Gamma$, the set of attributes used to create the private key.

Next we define the function f which transforms a set of attributes into an element of $\mathbb{G}$ as:

$$f(\Delta) = \prod_{x \in \Delta} h_x.$$

To decrypt, the algorithm will first do a pre-processing step on the private key. For each $i \in I$, it will compute the value

$$\hat{D}_i = D_i \cdot \prod_{x \in \Delta/\rho(i)} Q_{i,x} = g^{\lambda_i} f(\Delta)^{r_i}.$$

Next, the algorithm will do a pre-processing step on the ciphertext by computing the value

$$L = \prod_{x \in \Delta} C_x = \prod_{x \in \Delta} h_x^s = f(\Delta)^s.$$

The algorithm now recovers the value $e(g,g)^{\alpha s}$ by computing

$$e(\hat{C}, \prod_{i \in I} \hat{D}_i^{\omega_i}) / e(\prod_{i \in I} R_i^{\omega_i}, L) =$$
$$e(g^s, \prod_{i \in I} g^{\lambda_i \omega_i} f(\Delta)^{r_i \omega_i}) / e(\prod_{i \in I} g^{r_i \omega_i}, f(\Delta)^s) =$$
$$e(g,g)^{\alpha s} \cdot e(g, f(\Delta))^{s \sum_{i \in I} r_i \omega_i} / e(g, f(\Delta))^{s \sum_{i \in I} r_i \omega_i} = e(g,g)^{\alpha s}.$$

The decryption algorithm can then divide out this value from C and obtain the message M. The decryption algorithm requires the computation of only two pairing operations.

3.2 Efficiency and Tradeoffs

The main feature of the above scheme is that decryption only requires two pairings. While decryption also requires two exponentiations per row used, if the LSSS is derived from an AND/OR tree then the exponents w_i will be 1. (That is, they will be either 0 or 1, but the $w_i = 0$ rows should not be used.) Thus, decryption can be very fast. There are two tradeoffs:

1. The private key size and generation time blows up by roughly a factor of $|\Gamma|$ compared to GPSW, where Γ is the set of distinct attributes used in making the key.
2. Decryption reduces the number of pairings, but requires modular multiplications of roughly a factor of $|\Delta|$ compared to GPSW, where Δ is the set of distinct attributes used in decryption.

Thus, while there is a blow-up, this increase is tied only to the number of distinct attributes "touched" by the corresponding operation, and not by a global bound. Depending on the application, one should take into consideration whether the blow up in key size is worth it. Moreover, the decryption time could actually increase over GPSW [13] once Δ becomes sufficiently large. However, it would have to be *so large* that 2 pairings plus $|I| \cdot |\Delta|$ multiplications dominates $|\Delta|$ pairings and $|I|$ multiplications. As one benchmark, it required 8.22ms to compute a pairing for a BN256 curve with the RELIC library on a modern PC while roughly 0.0034ms to compute a modular multiplication. Thus, in a setting where $|I| = |\Delta|$ (the number of rows of the access matrix touched during decryption is the same as the number of distinct attributes touched), the decryption algorithm would need to touch over 2416 distinct attributes before GPSW would be faster. Should this occur, however, it is worth noting that the above private keys actually contain a GPSW key; thus, if this threshold was ever reached, one could revert back to doing GPSW decryption.

We summarize the comparison of GPSW and our construction in Figure **??**. In Section 5, we will provide a generalized construction for finer-grained tradeoff optimization.

3.3 Large Universe Realizations

The construction in Section 3.1 can be transformed so that *any* string can be a valid attribute; that is, $U = \{0,1\}^*$, as follows: Assume that all parties have access to a hash function $H : \{0,1\}^* \to \mathbb{G}$, which will be treated as a random oracle. Remove the values $h_1, \ldots, h_{|U|}$ from the public parameters PK. For any attribute $x \in \{0,1\}^*$, define the value $h_x = H(x)$. Otherwise, follow the construction as written. Thus, the efficiency is the same, modulo additional hash function evaluations. Regarding the proof of security, let q be the maximum number of unique queries made to the random oracle. Then, this large universe construction is selectively, CPA-secure under the Decisional q-BDHE assumption in the random oracle model. The proof will follow the outline of that in Section 4 except that Setup no longer outputs any h_x values and instead $\mathcal{B}$ must simulate the random oracle as follows. It should initialize an empty table T_{RO} at the beginning of the experiment. On each query for attribute x to the random oracle, $\mathcal{B}$ should first look to see if x is in T_{RO} and if so, return the value associated with it.

If x is not in T_{RO}, $\mathcal{B}$ creates a new table entry (x, i, h_x) for it as follows. Let i be the number of unique attributes queried to the random oracle (including x) at the time of this query. Let z_x be a random value in $\mathbb{Z}_p$. Then set

$$h_x := \begin{cases} g^{z_x} & \text{if } x \in S^*; \\ g^{z_x} g^{a^i} & \text{if } x \notin S^*. \end{cases}$$

An interesting question is whether one can achieve fast decryption for a large universe in the standard model. Lewko and Waters [16] gave a large universe construction for KP-ABE in the standard model. However, their technique requires that each attribute in the ciphertext have some "local" randomness associated with it. This does not work with our methods here which leverage the fact that there is only one random exponent that propagates through the ciphertext.

3.4 Short Ciphertext Realizations

In Section 3.1, we focused on optimizing decryption time. For some applications where bandwidth or storage space is a practical concern, one might prefer to optimize on ciphertext size. In the small universe construction, we can compress the ciphertext into only three group elements (as opposed to $2 + |S|$ group elements in Section 3.1) plus the description of the attribute set S. The main tradeoff is that private key sizes must scale by a factor of the size $|U|$ of the universe (compared to GPSW [13]), as opposed to scaling by only Γ as above.

We now sketch the main idea. During encryption, instead of including the set $\{C_x = h_x^s\}_{x \in S}$ in the ciphertext, it now includes the product of these values,

the "aggregate" $\prod_{x \in S} C_x = f(S)^s$. Thus, the ciphertext contains three group elements of the form:

$$C = M \cdot e(g,g)^{\alpha s},\ \hat{C} = g^s,\ \prod_{x \in S} C_x = f(S)^s.$$

When generating the private key, replace Γ with U; that is, instead of only having "helper values" $Q_{i,x}$ for attributes x used in the access matrix, one now must include helpers for any attribute in the universe. This will allow the decryptor to handle an aggregate of any set of attributes. Thus, following the setup as before, the private keys are of the form:

$$\begin{aligned} \text{PK}, \quad & (D_1 = g^{\lambda_1} \cdot h_{\rho(1)}^{r_1},\ R_1 = g^{r_1},\ \forall d \in U/\rho(1),\ Q_{1,d} = h_d^{r_1}),\ \ldots\ , \\ & (D_\ell = g^{\lambda_\ell} \cdot h_{\rho(\ell)}^{r_\ell},\ R_\ell = g^{r_\ell},\ \forall d \in U/\rho(\ell),\ Q_{\ell,d} = h_d^{r_\ell}). \end{aligned}$$

The key size is proportional to $|U| \cdot \ell$, which is size of the attribute universe times the number of rows in the matrix. Due to the dependence on $|U|$, this aggregation unfortunately only works for small universes of attributes.

Finally, run the decryption algorithm as it is written, but understand that Δ will always be S due to the aggregate. This will increase the number of modular multiplications over Section 3.1, but not the number of pairings.

3.5 CP-ABE Variants

One might consider trying to apply these techniques in the CP-ABE setting [5, 12] by analogously modifying an "unrestricted" CP-ABE construction, such as Waters [20, Section 3]. A natural analogy would arise in a CP-ABE system where the ciphertext size and encryption time blows up by a factor of X, where X is the number of distinct attributes used in the ciphertext access structure. In some applications, one might consider this to be a less palatable tradeoff. That is, one might not be willing to increase the transmission costs (i.e., ciphertext size) and encryption time, even if it meant faster decryption times.

We note, however, that if one is willing to consider "bounded" systems, where a value k_{max} can be set system-wide as the maximum number of times a single attribute can appear in a particular formula (or access structure), then one can achieve fast decryption without an increase in ciphertext size or encryption time. One such example is the CP-ABE construction of Waters [20, Section 5]. The critical part of the decryption algorithm appears in that paper as

$$e(C', K) / \left(\prod_{i \in I} (e(C_i, L) \cdot e(C', K_{\rho(i)}))^{\omega_i} \right)$$

which seems to require a non-constant $(2|I|+1)$ pairings. However, this equation is identical to:

$$e(\prod_{i \in I} C_i^{-\omega_i}, L) \cdot e(C', K \prod_{i \in I} K_{\rho(i)}^{-\omega_i})$$

which requires only two pairings. The observation that this bounded scheme offers fast decryption was previously made in its Charm [1] implementation.

4 Proof of the Base Construction

Theorem 1 (Security of the Small Universe KP-ABE). *The KP-ABE scheme Π in Section 3.1 for attribute universe U is selectively secure against chosen-plaintext attacks under the Decisional $|U|$-BDHE assumption in $\mathbb{G}$.*

Proof. Let U be an attribute universe, where $|U|$ is a polynomial in 1^λ. For notational convenience, we will assume each of the $|U|$ attributes is a unique integer between 1 and $|U|$.[4] Next suppose there exists a PPT adversary $\mathcal{A}$ that causes the selective, CPA security experiment $\mathsf{KP\text{-}ABE\text{-}Exp}^{\mathsf{sel\text{-}CPA}}_{\mathcal{A},\Pi}(\lambda, U)$ to output 1 with non-negligible probability. Then, we can construct a PPT adversary $\mathcal{B}$ that violates the Decisional $|U|$-BDHE assumption in $\mathbb{G}$ as follows:

Init: $\mathcal{A}$ outputs a set S^* of attributes for the challenge ciphertext.
Setup: $\mathcal{B}$ receives the Decisional $|U|$-BDHE challenge input

$$(\mathbb{G}, p, g, g^s, g^a, \ldots, g^{(a^{|U|})}, g^{(a^{|U|+2})}, \ldots, g^{(a^{2|U|})}, P)$$

for security parameter λ. It chooses random $\alpha', z_1, \ldots, z_{|U|} \in \mathbb{Z}_p$ and sets $e(g,g)^\alpha := e(g,g)^{\alpha'} \cdot e(g^a, g^{a^{|U|}})$ (implicitly defining α as $(\alpha' + a^{|U|+1})$) and

$$\text{for } x \in [1, |U|], \text{ sets } h_x := \begin{cases} g^{z_x} & \text{if } x \in S^*; \\ g^{z_x} g^{a^x} & \text{if } x \notin S^*. \end{cases}$$

It sets the public parameters PK as $(\mathbb{G}, p, g, e(g,g)^\alpha, h_1, \ldots, h_{|U|})$ and sends them to $\mathcal{A}$. Note that all parameters are well distributed due to the α' and z_x values.

Phase 1: $\mathcal{B}$ initializes an empty table T, an empty set D and a counter $j = 0$. $\mathcal{B}$ responds to $\mathcal{A}$'s queries as follows:

1. Create($\mathbb{A}$): $\mathcal{B}$ sets $j := j + 1$. It parses $\mathbb{A}$ as (W, ρ), where W is an $\ell \times n$ matrix. $\mathcal{B}$ will now work in two steps. First, it will create a valid private key, but not necessarily a well-distributed one. Then, it will re-randomize the key to ensure it is well distributed.

Let K be the set of rows where the attributes are in S^* (i.e., for $i \in K, \rho(i) \in S^*$.) and K' be the rows where attributes are not in S^* (i.e., $K' = [1, \ell]/K$). Define an n dimensional vector $\boldsymbol{v}$ over $\mathbb{Z}_p$. Let $v_1 = 1$ (i.e., first element of $\boldsymbol{v}$ is 1) and for all $i \in K, \boldsymbol{v} \cdot W_i = 0$. (Here W_i is the n-dimensional vector that is row i of the matrix W.) To see that this is well-defined, consider that since S^* does not satisfy W, it must be the case that $(1, 0, 0..., 0)$ is *not* in the span of rows W_i for $i \in K$. It then follows from linear algebra that such a vector $\boldsymbol{v}$ exists.

Next, $\mathcal{B}$ will make a private key for secret sharing with the vector $\alpha\boldsymbol{v}$ (i.e., the vector $\boldsymbol{v}$ with all the components scaled up by α.) This shares the secret

[4] A more proper notation would define an injective function $t()$ mapping attributes to integers from 1 to $|U|$ and then wherever we refer to an attribute x to instead refer to $t(x)$. However, this is more cumbersome.

α since $v_1 = 1$, although the key may not be well distributed. This will be addressed later through re-randomization. The shares λ_i for $i \in [1, \ell]$ are computed as $(\alpha \boldsymbol{v}) \cdot W_i$.
For $i \in K$, we have that all components are just the identity element (recall that the key is not re-randomized yet; we add the h_x components in shortly with the randomization): $D_i = R_i = H_{i,x} = g^0$ for $x \in \Gamma$ (recall Γ is the set of distinct attributes used in key generation). This follows because for all $i \in K, \lambda_i = 0$.
For $i \in K'$, we first compute $c_i = \boldsymbol{v} \cdot W_i$. Note that $\lambda_i = c_i \cdot \alpha = c_i \cdot (\alpha' + a^{|U|+1})$. To produce these key components, we need a cancellation technique. Set $R_i = g^{-c_i a^{(|U|+1)-\rho(i)}}$, which implicitly defines $r_i = -c_i \cdot a^{(|U|+1)-\rho(i)}$. This is computable from $\mathcal{B}$'s challenge input since $\rho(i)$ is between 1 and $|U|$. Set

$$\begin{aligned} D_i &= g^{c_i \alpha'} \cdot R_i^{z_i} \\ &= g^{c_i \alpha'} \cdot g^{-z_i c_i a^{(|U|+1)-\rho(i)}} \\ &= g^{c_i \alpha'} \cdot g^{c_i a^{|U|+1}} \cdot g^{-z_i c_i a^{(|U|+1)-\rho(i)}} \cdot g^{-c_i a^{|U|+1}} \\ &= g^{c_i \alpha} \cdot (g^{z_i})^{r_i} \cdot (g^{a^{\rho(i)}})^{r_i} \\ &= g^{c_i \alpha} \cdot h_{\rho(i)}^{r_i} \end{aligned}$$

Next, we turn to computing the helper values. For all $x \in \Gamma / \rho(i)$, set $Q_{i,x} = h_x^{r_i}$ by computing as follows:

$$h_x^{r_i} := \begin{cases} (g^{z_x})^{r_i} = g^{-z_x c_i a^{(|U|+1)-\rho(i)}} & \text{if } x \in S^*; \\ (g^{z_x})^{r_i} \cdot (g^{a^x})^{r_i} = g^{-z_x c_i a^{(|U|+1)-\rho(i)}} \cdot g^{-c_i a^{(|U|+1)-\rho(i)+x}} & \text{if } x \notin S^*. \end{cases}$$

This last part is computable since $\rho(i) \neq x$ for the helper values and recall that the z_x values were chosen by $\mathcal{B}$ during Setup.
At this point, $\mathcal{B}$ has constructed the components of a valid private key. Next, we give a public-key re-randomization algorithm that can be applied by $\mathcal{B}$ to *any* valid private key, before it sends the key to $\mathcal{A}$. To re-randomize, choose random $y_2, ..., y_n \in \mathbb{Z}_p$. Consider the vector $(0, y_2, y_3, ..., y_n)$. This will be used to secret share 0 to re-randomize the key. Let $\lambda_i' = (0, y_2, y_3, ..., y_n) \cdot W_i$ for $i \in [1, \ell]$.
For all $i \in [1, \ell]$, the first step to re-randomization is to let $D_i^{\#} := D_i \cdot g^{\lambda_i'}$. The next step is to re-randomize all the r_i values, which are used in all key components. To do this, choose a fresh $r_i' \in \mathbb{Z}_p$ and set

$$D_i' := D_i^{\#} \cdot h_{\rho(i)}^{r_i'} \; R_i' := R_i \cdot g^{r_i'} \; Q_{i,x}' := Q_{i,x} \cdot h_x^{r_i'}, \forall x \in \Gamma / \rho(i)$$

We claim that the above re-randomization procedure correctly re-randomizes any "valid" key. A valid key is one which is generated from some sharing of α, but not necessarily a well distributed one. The above algorithm propagates new random values $r_1', \ldots, r_\ell'$ completely through all key components, and then also generates a fresh secret sharing for α used in $D_1', \ldots, D_\ell'$. This

properly redistributes the only variable parts of the key. That is the distribution after applying this transformation to any valid key with policy (W, ρ) has the same distribution as a fresh key generated by running KeyGen for (W, ρ).

2. Corrupt(i): If there exists an ith entry in table T, then $\mathcal{B}$ obtains the entry $(i, \mathbb{A}, \mathrm{SK})$ and sets $D := D \cup \{\mathbb{A}\}$. It then sends SK to $\mathcal{A}$. If no such entry exists, then it returns $\perp$.

Challenge: $\mathcal{A}$ outputs two messages M_0, M_1 and $\mathcal{B}$ chooses a random bit b. $\mathcal{B}$ then constructs and sends to $\mathcal{A}$ the challenge ciphertext

$$\mathrm{CT}^* = (C^* := M_b \cdot P,\ C' := g^s,\ \forall x \in S^*, C_x^* := (g^s)^{z_x}).$$

Phase 2: $\mathcal{B}$ responds to $\mathcal{A}$'s queries in the same manner as in Phase 1, except that it refuses to answer any Corrupt query that would result in an access structure $\mathbb{A}$ which S^* satisfies being added to D.

Guess: Eventually, $\mathcal{A}$ outputs a bit b'. If $b = b'$, then $\mathcal{B}$ outputs 0 (guessing that $P = e(g, g)^{a^{|U|+1}s}$), else it outputs 1 (guessing that P is random.)

Thus, $\mathcal{B}$'s responses to $\mathcal{A}$ are distributed identically as in the $\mathsf{KP\text{-}ABE\text{-}Exp}^{\mathsf{sel\text{-}CPA}}_{\mathcal{A},\Pi}(\lambda, U)$ experiment. Whenever $\mathcal{A}$ causes the output of this experiment to be 1, $\mathcal{B}$ will also correctly answer its Decisional BDHE challenge.

5 Exploring a Spectrum of Efficiency Tradeoffs

We now focus on the tradeoff between private key size and decryption time. We generalize the construction ideas of the last section to give a spectrum of possible "unbounded" schemes, where GPSW is one extreme and Section 3 (longer keys, faster decryption) is the other. We do this in two steps. We first present a generalized decryption algorithm. We then show how the size of the private key scales depending on how one chooses to take advantage of this generalized decryption algorithm. We conclude with strategies for keeping both key size and decryption time low.

5.1 A Generalized Decryption Algorithm

To begin, we present a generalized decryption algorithm for the GPSW ciphertexts (which are the same as the encryption algorithm presented in Section 3.1). The main idea is to break Δ (the set of distinct attributes associated with the rows of the access matrix used in one chosen way to decrypt the ciphertext) into y disjoint subsets $\Delta_1, \Delta_2, \ldots, \Delta_y$. Recall that we defined the function f as

$$f(\Delta) = \prod_{x \in \Delta} h_x \in \mathbb{G}.$$

Further, let us establish the notation for a function w that (informally) takes in a attribute and outputs which set the attribute is in. More formally, let $w : \Delta \to [1, y]$ be a map such that $w(x) = j$ if and only if $x \in \Delta_j$.

The decryption algorithm then proceeds as follows. For each $i \in I$, it will compute the value

$$\hat{D}_i = D_i \cdot \prod_{x \in \Delta_{w(\rho(i))}/\rho(i)} Q_{i,x} = g^{\lambda_i} f(\Delta_{w(\rho(i))})^{r_i}.$$

Next, for each $j \in [1, y]$, it will compute the value

$$L_j = \prod_{x \in \Delta_j} C_x = \prod_{x \in \Delta_j} h_x^s = f(\Delta_j)^s.$$

The algorithm now recovers the value $e(g,g)^{\alpha s}$ by computing

$$e(\hat{C}, \prod_{i \in I} \hat{D}_i^{\omega_i}) / \left(\prod_{j=1}^{y} e(\prod_{i:\rho(i) \in \Delta_i} R_i^{\omega_i}, L_j) \right).$$

The decryption algorithm can then divide out this value from C and obtain the message M.

We observe that this will take $(1 + y)$ pairings and roughly $|\Delta_1|^2 + |\Delta_2|^2 + \cdots + |\Delta_y|^2$ modular multiplications. (Recall that the ω_i values will be either 0 or 1 when the access structure is a boolean formula, so no exponentiations come into play in this case.) Thus, when $y = 1$, we have the scheme from Section 3.1 (which can be trivially extended to large universes in the random oracle model as shown in Section 3.3) and when $y = |\Delta|$, this corresponds to GPSW.

5.2 Reducing Private Key Overhead

At this point, the private key still contains "helper" values such that each of the ℓ rows has helpers for all other attributes in Δ. However, we can reduce the size of the private key by eliminating all helper values $Q_{i,x}$ where where attribute $\rho(i) \in \Delta_d$, attribute $x \in \Delta_{d'}$ and $d \neq d'$, i.e., where the two attributes were separated into distinct subsets. This is because only helpers *within* subsets will be used in the generalized decryption algorithm. We note that the security of the base system trivially implies security of this system, since for each private key in this reduced setting the components given out are a strict subset of the private key components in the base system.

5.3 Different Tradeoff Strategies

We now discuss how one might take advantage of the generalized algorithm and corresponding private key reduction. The choice of y and the subsets $\Delta_1, \ldots, \Delta_y$ is critical to the decryption *performance of a particular user*. Roughly, one expects decryption time to increase with y, but the size of the private key to decrease as the size of each Δ_i decreases. However, a nice feature of this approach is that each user can tune their own performance based on how they think they are likely to use their private key. A user might choose to retain her entire private key on her PC, but upload only a portion of her private key to her mobile device (where secure storage may be more limited.)

1. *Group attributes by expected ciphertext attributes.* Group together any attributes that are likely to appear together in a ciphertext, such as attributes relating to a certain work project, activity, role or time period. For instance, one might group together the attributes "cryptography", "encryption", and "pairings" and form a distinct group for the attributes "audubon", "peregrine falcon", "glaucous gull". Of course, the subsets of Δ need not be distinct[5] and it could be more efficient to place an attribute into two or more groups, but this should be done with care or the decryption time will increase without reducing the private key size.
2. *Group attributes by observing the private key.* It may be possible to deduce from the access structure which attributes are likely to be used together during decryption. For instance, suppose the structure is a formula and the only time that attributes A and B appear, they appear as "A AND B". Then clearly one should place A and B into the same group Δ_i.
3. *Break into y equal sized subsets of attributes.* One could also try the simple approach of choosing a y and randomly creating y equal-sized subsets. In many practical applications, the average overhead incurred on future ciphertexts would be dependent on the overhead from past ciphertexts, so one could try a random setting and then observe performance.
4. *Benchmark and set experimentally.* One can also imagine starting with any combination of the three above techniques and then applying machine learning tools to evolve to a good balance point for any particular user.

We leave an evaluation of these strategies and their performance as an interesting open problem.

Acknowledgments

The authors thank Matthew Green for helpful input and discussions and the anonymous reviewers for helpful comments. This work was performed while the authors were at Zeutro, LLC.

References

1. J Ayo Akinyele, Gary Belvin, Christina Garman, Matthew Pagano, Michael Rushanan, Paul Martin, Ian Miers, Matthew Green, and Avi Rubin. Charm: A tool for rapid cryptographic prototyping. Available from `http://www.charm-crypto.com/`, 2012.
2. Nuttapong Attrapadung, Javier Herranz, Fabien Laguillaumie, Benoît Libert, Elie de Panafieu, and Carla Ràfols. Attribute-based encryption schemes with constant-size ciphertexts. *Theor. Comput. Sci.*, 422:15–38, 2012.
3. Nuttapong Attrapadung, Benoît Libert, and Elie de Panafieu. Expressive key-policy attribute-based encryption with constant-size ciphertexts. In *Public Key Cryptography*, pages 90–108, 2011.

[5] If the subsets are not distinct, then one needs to generalize the function w to output a subset of indices.

4. Amos Beimel. *Secure Schemes for Secret Sharing and Key Distribution*. PhD thesis, Israel Institute of Technology, Technion, Haifa, Israel, 1996.
5. John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *IEEE Symposium on Security and Privacy*, pages 321–334, 2007.
6. Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In *EUROCRYPT*, pages 440–456, 2005.
7. Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In *CRYPTO*, pages 213–229, 2001.
8. Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In *CRYPTO*, pages 258–275, 2005.
9. Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In *EUROCRYPT*, pages 255–271, 2003.
10. Clifford Cocks. An identity based encryption scheme based on quadratic residues. In *IMA Int. Conf.*, pages 360–363, 2001.
11. Craig Gentry and Alice Silverberg. Hierarchical id-based cryptography. In *ASIACRYPT*, pages 548–566, 2002.
12. Vipul Goyal, Abishek Jain, Omkant Pandey, and Amit Sahai. Bounded ciphertext policy attribute-based encryption. In *ICALP*, 2008.
13. Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM Conference on Computer and Communications Security*, pages 89–98, 2006.
14. Jeremy Horwitz and Ben Lynn. Toward hierarchical identity-based encryption. In *EUROCRYPT*, pages 466–481, 2002.
15. Allison Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, 2010.
16. Allison B. Lewko and Brent Waters. Unbounded HIBE and attribute-based encryption. In *EUROCRYPT*, pages 547–567, 2011.
17. Tatsuaki Okamoto and Katsuyuki Takashima. Fully secure functional encryption with general relations from the decisional linear assumption. In *CRYPTO*, pages 191–208, 2010.
18. Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, pages 457–473, 2005.
19. Adi Shamir. Identity-based cryptosystems and signature schemes. In *CRYPTO*, pages 47–53, 1984.
20. Brent Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In *PKC*, pages 53–70, 2011.

A Access Structures and Notation

A.1 Access Structures

Definition 1 (Access Structure [4]) *Let* $\{P_1, P_2, \ldots, P_n\}$ *be a set of parties. A collection* $\mathbb{A} \subseteq 2^{\{P_1, P_2, \ldots, P_n\}}$ *is monotone if* $\forall B, C$: if $B \in \mathbb{A}$ *and* $B \subseteq C$ *then* $C \in \mathbb{A}$. *An* access structure *(respectively, monotone access structure) is a collection (resp., monotone collection)* $\mathbb{A}$ *of non-empty subsets of* $\{P_1, P_2, \ldots, P_n\}$, *i.e.,* $\mathbb{A} \subseteq 2^{\{P_1, P_2, \ldots, P_n\}} \setminus \{\emptyset\}$. *The sets in* $\mathbb{A}$ *are called the* authorized sets, *and the sets not in* $\mathbb{A}$ *are called the* unauthorized sets.

In our context, the role of the parties is taken by the attributes. Thus, the access structure $\mathbb{A}$ will contain the authorized sets of attributes. We restrict our attention to monotone access structures. However, it is also possible to (inefficiently) realize general access structures using our techniques by defining the "not" of an attribute as a separate attribute altogether. Thus, the number of attributes in the system will be doubled. From now on, unless stated otherwise, by an access structure we mean a monotone access structure.

A.2 Linear Secret Sharing Schemes

The construction will use linear secret sharing schemes, as slightly adapted from Beimel [4]:

Definition 2 (Linear Secret-Sharing Schemes (LSSS)) *A secret-sharing scheme Π over a set of parties $\mathcal{P}$ is called* linear *(over $\mathbb{Z}_p$) if*

1. *The shares of the parties form a vector over $\mathbb{Z}_p$.*
2. *There exists a matrix M with ℓ rows and n columns called the share-generating matrix for Π. There exists a function ρ which maps each row of the matrix to an associated party. That is for $i = 1, \ldots, \ell$, the value $\rho(i)$ is the party associated with row i. When we consider the column vector $v = (s, r_2, \ldots, r_n)$, where $s \in \mathbb{Z}_p$ is the secret to be shared, and $r_2, \ldots, r_n \in \mathbb{Z}_p$ are randomly chosen, then Mv is the vector of ℓ shares of the secret s according to Π. The share $(Mv)_i$ belongs to party $\rho(i)$.*

It is shown in [4] that every linear secret sharing-scheme according to the above definition also enjoys the *linear reconstruction* property, defined as follows: Suppose that Π is an LSSS for the access structure $\mathbb{A}$. Let $S \in \mathbb{A}$ be any authorized set, and let $I \subseteq \{1, 2, \ldots, \ell\}$ be defined as $I = \{i : \rho(i) \in S\}$. Then, there exist constants $\{\omega_i \in \mathbb{Z}_p\}_{i \in I}$ such that, if $\{\lambda_i\}$ are valid shares of any secret s according to Π, then $\sum_{i \in I} \omega_i \lambda_i = s$. It is shown in [4] that these constants $\{\omega_i\}$ can be found in time polynomial in the size of the share-generating matrix M.

Like any secret sharing scheme, it has the property that for any unauthorized set $S \notin \mathbb{A}$, the secret s should be information theoretically hidden from the parties in S.

Note on Convention. We use the convention that vector $(1, 0, 0, \ldots, 0)$ is the "target" vector for any linear secret sharing scheme. For any satisfying set of rows I in M, we will have that the target vector is in the span of I.

For any unauthorized set of rows I the target vector is not in the span of the rows of the set I. Moreover, there will exist a vector w such that $w \cdot (1, 0, 0 \ldots, 0) = -1$ and $w \cdot M_i = 0$ for all $i \in I$.

Using Access Trees. Some prior ABE works (e.g., [13]) described access formulas in terms of binary trees. Using standard techniques [4] one can convert any monotonic boolean formula into an LSSS representation. An access tree of ℓ nodes will result in an LSSS matrix of ℓ rows.